VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ANALÝZA ABSTRAKTNÝCH SYNTAKTICKÝCH STROMOV PRE PODPORU VÝUKY JAZYKA PYTHON

ABSTRACT SYNTAX TREE ANALYSIS FOR PYTHON TEACHING SUPPORT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB FÁBER**
AUTHOR

VEDOUCÍ PRÁCE **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Fáber Jakub**
Obor: Informační technologie
Téma: **Analýza abstraktních syntaktických stromů pro podporu výuky jazyka Python**
Abstract Syntax Tree Analysis for Python Teaching Support
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se se systémy typu Automatic Tutor se zaměřením na výuku pokročilých technik programování a s metodami analýzy kódu na syntaktické a sémantické úrovni.
2. Získejte a zpracujte přehled nástrojů pro odhalování nedostatků v kódu dynamických jazyků a možností doporučování lepších postupů z hlediska výkonnosti a čistoty kódu.
3. Navrhněte a implementujte systém, který bude na základě rozboru kódu analyzovat programová řešení v Pythonu a navrhovat zlepšení.
4. Vyhodnoťte vytvořený systém na sadě příkladů relevantních pro předmět Skriptovací jazyky.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle domluvy s vedoucím.

Pro udělení zápočtu za první semestr je požadováno:

- Funkční prototyp

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cieľom tejto práce je vytvorenie analyzátora za účelom kontroly programových riešení v jazyku Pythonu a navrhovanie ich zlepšení z hľadiska kvality a čistoty kódu. Pre najlepšie výsledky sa využíva kombinácia statickej a dynamickej analýzy. Tento systém je implementovaný vo forme tutor systému a využíva Docker kontajnery na bezpečné testovanie a analýzu v izolovanom prostredí. Na komunikáciu s úžívateľom je navrhnuté interaktívne prostredie s použitím moderných technológii ako je Material Design.

Abstract

The goal of this thesis is to create an analyzer system to check assignment solutions and to suggest improvements for them in terms of code quality and cleanliness. For the best results, a combination of static and dynamic analysis techniques is used. This system is implemented in the form of tutoring system and uses Docker containers for safe testing and analysis in an isolated environment. Interactive environment for communication with users is created with modern technologies such as Material Design.

Kľúčové slová

python, učenie, analýza, AST, tutor, systém, aplikácia

Keywords

python, learning, analysis, AST, tutor, system, application

Citácia

FÁBER, Jakub. *Analýza abstraktných syntaktických stromov pre podporu výuky jazyka Python.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Smrž Pavel.

Analýza abstraktných syntaktických stromov pre podporu výuky jazyka Python

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána docenta Pavla Smrža. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Jakub Fáber
18. mája 2017

Poďakovanie

Chcel by som v tejto sekcii poďakovať pánovi docentovi Pavlovi Smržovi za poskytnutý čas a cenné rady pri riešení bakalárskej práce.

Obsah

1 Úvod **3**

2 Motivácia na vytvorenie systému *Python Tutor* **4**

3 Teoretický základ **6**
3.1 Proces spustenia kódu v CPython 6
3.2 Metódy analýzy 9
3.2.1 Statická analýza 9
3.2.2 Dynamická analýza 9

4 Porovnanie existujících riešení **10**
4.1 Tutoring systémy 10
4.2 Vizualizačné nástroje 11
4.3 Analyzačné nástroje 11

5 Použité technológie **13**
5.1 Python 13
5.2 Astroid 13
5.3 XML a XPath 14
5.4 Flask 14
5.5 Docker 14
5.6 AngularJS 15
5.7 Code Mirror 15

6 Návrh riešenia a implementácia **16**
6.1 Moduly systému 17
6.1.1 Client 17
6.1.2 Server 21
6.1.3 Databáza 21
6.1.4 Runner 21
6.1.5 Tester 22
6.1.6 Collector 22
6.1.7 Analyzer 23

7 Testovanie **27**
7.1 Analýza úlohy č. 1 28
7.2 Analýza úlohy č. 2 29
7.3 Analýza úlohy č. 3 30

8 Záver **31**

Literatúra **32**

Prílohy **35**

A Obrázky alternatívnych riešení **36**

B Náležitosti potrebné pre nahranie novej úlohy do systému **38**

C Obsah súborov na disku **40**

Kapitola 1

Úvod

V súčasnosti získava Python na obľúbenosti, ako u úpných začiatočníkov, tak aj u užívateľov so skúsenosťami z iných jazykov. Jeho výhodou je jednoduchosť a čitateľnosť, z tohto dôvodu sa využíva na mnohých univerzitách a kurzoch programovania. Znalosť Pythonu otvára možností v nových oboroch ako je napríklad Strojové učenie.

Tato práca popisuje tvorbu systému na učenie jazyka Python formou Automatic Tutor so zameraním na výuku pokročilých techník programovania a s metódami analýzy kódu na syntaktické, sémantické úrovni a analýzu konkurenčných riešení v oblasti tutoringu spojeného s analýzou zdrojového kódu. Takýto typ systému bol vybraný za účelom pomoci študentom s učením nového jazyka a využitím lepších postupov z hľadiska výkonnosti a čistoty kódu.

Dôvodom tvorby práce sú nedostatky, ktoré majú existujúce riešenia čo sa týka spojenia tutoringu a zároveň analýzy kódu. Za hlavný problém, s ktorým sa dá pri používaní aktuálnych riešení stretnúť, je že existujú systémy buď len na tutoring pomocou úloh a testovaním voči riešeniam, alebo analýzou kódu, prípadne len ako vizualizátory jazyka Python. Podrobnou analýzou konkurenčných riešení sa zaoberá kapitola 4.

Cieľom tejto práce je spojenie automatického tutoring systému pre jazyk Python, vrátane statickej a dynamickej analýzy zdrojového kódu za účelom efektívnejšej výuky študenta s okamžitou spätnou väzbou. Hlavnou časťou je analyzér, ktorého funcionalita bude overená na sade úloh z predmetu Skriptovacie jazyky. Kapitola 7 sa venuje rozboru výsledkov analýz jednotlivých úloh.

V práci sú taktiež vysvetlené základné princípy analýzy kódu u dynamicky typovaných jazykov.

Kapitola 2

Motivácia na vytvorenie systému *Python Tutor*

V posledných rokoch získava jazyk Python trakci ako preferovaný jazyk pre kurzy CS[1] na mnohých univerzitách. Napríklad dva z najväčších CS oddelení – MIT[2] a UC[3] Barkeley – obe priniesli svoje učebné osnovy CS od jazyka Scheme k Pythonu. Michiganská štátna univerzita prešla z C++ k Pythonu a predložili empirické dôkazy, že noví študenti boli rovnako pripravení na pokročilé kurzy CS vyučovaných v iných programovacích jazykoch [30].

Pri učení nových jazykov sa študenti často snažia aplikovať už naučené idiomy a vychádzajú z ich dosavadných skúsenosti. Každý jazyk má ale vlasné idiomy a odporúčané postupy, ktoré sa vybudovali časom a aktívnym používaním daného jazyka. Naučenie syntaxe a práce so štandardnou knihovnou, prípadne s knihovnami tretích strán ich to pravdepodne nenaučí.

Jednou z možností získavania praktických znalostí je *Pair programming* [4]. Menej skúsení programátori, ktorí sa učia nové idiomy a vlastnosti jazyka od mentorov, rychlešie rovíjajú svoje schopnosti vďaka okamžitej odozve [29].

Ďalšou z možností ako sa môže začínajúci programátor učiť je pomocou online kurzov, napríklad MOOC[5]. V súšasnej dobe priťahujú MOOC stále viac pozornosti [14]. Aj napriek tomu, že len 10% zo zapísaných do kurzov sa dostane na konec, zapisuje sa ich priemerne až 43000 na jeden kurz [13]. Vzhľadom k veľkému záujmu nie je v silách učiteľa venovať sa študentom osobne a analyzovať im zdrojové kódy a doporučovať lepšie postupy. Je to časovo aj fyzicky nezvládnuteľné.

Tutoring system je systém, ktorého cieľom je poskytnúť veľkému možstvu študentov prístup k kvalitnému vzdelávaniu, a to efektívnym spôsobom pomocou rôznych výpočtových technológií [33]. Jeho úlohou je poskytnúť im okamžitú spätnú väzbu, zvyčajne bez potreby zásahu učiteľa. V dnešnej dobe sa tutoring systémy snažia úlohu učiteľa alebo asistenta učiteľa uľahčiť a čoraz viac automatizujú pedagogické funkcie, ako je napríklad generovanie problémov, výber problémov a generovanie spätnej väzby [34].

V súčasnosti existujú systémy typu tutor, ktoré poskytujú úlohy na vyriešenie. Mnohé z nich sa nešpecializujú na jeden konkrétny jazyk, ale snažia sa mať najväčšiu podporu

[1] Computer Science
[2] Massachusetts Institute of Technology
[3] University of California
[4] Párové programovanie: http://www.knesl.com/jak-funguje-parove-programovani
[5] Masívne online kurzy

vstupných jazykov. Iné sa zameriavajú priamo na Python, ale sú hlavne na podporu učenia jazyka alebo jeho syntaxe. Žiaden z nich ale nepoužíva analýzu za účelom poskytnutia spätnej väzby ohľadom kvality kódu, poprípade jeho idiomatickosti. Podrobnou analýzou konkurenčných riešení sa zaoberá kapitola 4.

Začínajúci programátori si pri výberu prvého jazyka stále častejšie volia Python [21]. Medzi jeho najväčšie výhody patria:

- *čitateľnosť kódu* – Python má filozofiu, ktorá kladie dôraz na čitateľnosť kódu (hlavne pomocou odsadenia bielych znakov pre vymedzenie blokov kódu) a syntaxe. Tá umožňuje programátorom vyjadrovať koncepty v niekoľkých riadkoch kódu, ako to nie je možné v jazykoch C++ alebo Java [15]. Jazyk poskytuje konštrukcie určené na zapisovanie zrozumitelnych programov v malom aj veľkom rozsahu. Jeho jednoduchá a ľahko naučiteľná syntax znižuje bariéru pre začiatočníkov v programovaní.

- *dokumentácia a literatúra* – Pri učení sa nového jazyka je základným kameňom dobrá a detailná dokumentácia[6]. Štandardná dokumentácia Pythonu taktiež obsahuje kvalitný tutoriál[7] pre začiatočníkov.

- *modulárnosť* – Python podporuje moduly a balíky, čo prispieva k modularite programov a znovupoužiti ich kódu. Repozitár PyPI[8] v súčasnosti obsahuje okolo 108-tisíc balíkov pripravených na použitie. Inštalácia nových balíkov je jednoduchá, vo väčšine prípadov stačí príkaz `pip install PACKAGE_NAME`.

- *aktuálnosť a používanosť* – Jeho používanosť sa každoročne zvyšuje. Python má silné zastúpenie i v nových disciplínach ako je strojové učenie, Data Science, Big Data a automatizácia. Jazyk sa neustále rozvíja a obohacuje o nové myšlienky a inovácie. Posledná verzia interpretu Cython vo verzií 3.6 priniesla podporu *asynchrónneho kódu*[9] a *gradual typing*[10].

Python je vysoko-úrovňový interpretovaný, objektovo orientovaný programovací jazyk. Jeho vysoko-úrovňové dátové štruktúry (napr. zoznamy, sety, reťazce) v kombinácii s dynamickým typovaním ho robia veľmi atraktívny pre Rapid Application Development, ako skriptovací jazyk, alebo lepidlo za účelom spojenia súčasných komponentov do celku [20].

[6] Python dokumentácia: https://docs.python.org/3/
[7] Python tutoriál: http://docs.python.org/3/tutorial/
[8] Python Package index: https://pypi.python.org/pypi
[9] Generátory a AsyncIO: http://naucse.python.cz/lessons/intro/async/
[10] Gradual typing: http://homes.soic.indiana.edu/jsiek/what-is-gradual-typing/

Kapitola 3

Teoretický základ

Najprv si stručne vysvetlíme, ako funguje interpret jazyka Python, cez aké fázy musí zdrojový kód prejsť, od parsovania po interpretáciu. Následne sa pozrieme na techniky, ktoré sa aktuálne používajú na analýzu kódu. Tieto pojmy a procesy sú dôležité pre pochopenie procesu vytvárania bakalárskej práce.

3.1 Proces spustenia kódu v CPython

CPython je referenčná implementácia programovacieho jazyka Pythonu.

Zdrojový kód má v dnešnej dobe podobu reťazca znakov, ktoré sa postupne pretransformujú do výsledneho spustiteľného programu. Pred každým spustením skriptu sa v Pythone musia vykonať 4 fázy (vysvetlené v tejto kapitole), a ak sa počas ich vykonávania nevyskytne žiadna chyba, tak až potom môžeme vidieť výsledok nášho programu.

3.1.1 Lexikálna analýza

Na začiatok si vysvetlíme základné termíny, ktoré sa budú vyskytovať v tejto sekcii.

- *Lexémy* sú reťazece znakov pochádzajuce zo vstupného súboru, ktoré zodpovedajú istému vzoru. Lexém je rozpoznávaný lexikálnym analyzátorom. Je syntaktickou jednotkou najnižšej úrovne v programovacom jazyku. Označujú sa aj ako slová interpunkcia programovacieho jazyka [5].
- *Tokeny* reprezentujú štruktúru tvorenú názvom tokenu a voliteľnými parametrami (atribútmi). Názov tokenu predstavuje abstraktný symbol slúžiaci k identifikácii druhu lexikálnej jednotky (napr. identifikátor, kľúčové slovo). Tokeny sú vstupné jednotky pre syntaktický analyzátor.
- *Vzory* sú popisy istého tvaru, ktoré môžu lexémy nadobudnúť.

V tabuľke 3.1 je prehľadne zobrazený vzťah medzi *lexémom*, *tokenom* a *vzorom*.

Spracovávanie Python programu začína v parseri[1]. Lexikálna analyza je proces konvertovania postupnosti znakov do postuponsti tokenov.

[1] Lexikálna analýza - Parser: https://docs.python.org/3/reference/lexical_analysis.html

Lexém	*Token*	*Vzor*
name, calc_fib, x_1	**ID**	znak nasledovaný znakmi alebo číslami
if	**IF**	reťazec znakov `i`, `f`
<=, ==	**COMPARSION**	<, >, <=, >=, == alebo !=
else	**ELSE**	reťazec znakov `e`, `l`, `s`, `e`
2.7154, -1, 66.6e-4	**NUMBER**	číselná konštanta
"hello world"	**literal**	ľubovolné znaky ohraničené " alebo ’

Tabuľka 3.1: Príklady tokenov jazyka Python. Inšpirované z [24].

```
digit          ::=  "0"..."9"
digitpart      ::=  digit (["_"] digit)*
fraction       ::=  "." digitpart
exponent       ::=  ("e" | "E") ["+" | "-"] digitpart
pointfloat     ::=  [digitpart] fraction | digitpart "."
floatnumber    ::=  pointfloat | exponentfloat
imagnumber     ::=  (floatnumber | digitpart) ("j" | "J")
```

Ukážka 3.1: Časť špecifikácie lexémov

3.1.2 Parsovanie

Vstupom v tomto kroku je postupnosť *tokenov* z prvého bodu, pomocou ktorých sa následne vytvorí *abstraktný syntaktický strom*, čo je minimálna stromová reprezentácia zdrojového kódu, zbavená o komentáre a o elementy, ktoré sa dajú spätne získať zo stromu [23]. Ten je vytváraný podľa gramatiky jazyka, v ukážke 3.2 je uvedená časť gramatiky jazyk Python.

Strom vyvorený v tomto kroku sa intenzívne používa počas sémantickej analýzy, kde sa kontroluje správne použitie prvkov programu a jazyka. Taktiež sa v tomto kroku generuje tabuľka symbolov[2] založená na tomto strome.

Väčšina syntaktických analyzátorov s ním pracuje za účelom odhlania chýb v programe, a keďže je to klasická datová štruktúra typu strom[3], je v ňom pohyb celkom triviálny.

Podrobnejšie sa tomu venuje sekcia 3.2.1 a 6.1.7.

```
star_expr: '*' expr
expr: xor_expr ('|' xor_expr)*
xor_expr: and_expr ('^' and_expr)*
and_expr: shift_expr ('&' shift_expr)*
shift_expr: arith_expr (('<<'|'>>') arith_expr)*
arith_expr: term (('+'|'-') term)*
term: factor (('*'|'@'|'/'|'%'|'//') factor)*
factor: ('+'|'-'|'~') factor | power
power: atom_expr ['**' factor]
```

Ukážka 3.2: Časť z gramatiky jazyka Python

[2]Tabuľka symbolov: https://docs.python.org/3.6/library/symtable.html
[3]Dátová štruktúra strom: http://voho.eu/wiki/datova-struktura-strom/

3.1.3 Kompilácia

Nie je to veľmi známa informácia, ale aj napriek tomu, že je Python interpretovaný dynamický jazyk, prebieha v ňom kompilácia. Vstupom kompilátoru je *abstraktný syntaktický strom* z predchádzajúceho kroku, ktorý sa prekonvertuje do *kódoveho objektu* (code object). Aj keď tieto objekty predstavujú nejaký kus bajtkódu, nie sú sami o sebe priamo spustiteľné. Pre ich vykonanie musí interpret použiť funkciu `exec` alebo `eval` [7].

3.1.4 Interpretácia

Virtuálny stroj jazyka Python pracuje na strojovo nezávislom *stack-based* prúde (stream) bajtkódu, ktorý riadi vykonávanie programu. *Bajtkód* je vnútorná reprezentácia programu z tretieho kroku. Ukážka 3.4 ukazuje bajtkódovú reprezentáciu funkcie 3.3 v binárnom formáte. Modul `dis` zo štandardnej knihovny dokáže demontovať (disassemble) bajtkód interpretu a reprezentovať ho ako sekvenciu čitateľných *op-kódov* (opcodes). Tieto kódy využívajú zásobník ako dátovú štruktúru na uchovávanie stavu, ako aj komunikačný nástroj pri predávaní parametrov. V ukážke 3.5 môžeme vidieť *op-kód* `LOAD_FAST`, ktorý inštruuje interpret, aby uložil hodnotu premennej `a` v aktuálnom contexte a uložil ju na zásobník. Komplementána inštrukcia `STORE_FAST` slúží na jej odobratie zo zásobníka.

```
def rectangle_circ(a, b):
    return 2 * (a + b)
```

Ukážka 3.3: Funcia v jazyke Python

```
>>> rectangle_circ.__code__.co_code
b'd\x01|\x00|\x01\x17\x00\x14\x00S\x00'
```

Ukážka 3.4: Bajtkódová reprezentácia funkcie 3.3 v binárnej forme

```
0  LOAD_CONST               1  (2)
2  LOAD_FAST                0  (a)
4  LOAD_FAST                1  (b)
6  BINARY_ADD
8  BINARY_MULTIPLY
10 RETURN_VALUE
```

Ukážka 3.5: Bajtkódová reprezentácia funkcie 3.3 v čitateľnej forme

3.2 Metódy analýzy

V súčasnosti sa ako najčastejším vstupom do analyzérov jazyka Python používajú *abstrakné syntatktické stromy*. Pomocou nich je možné písať programy, ktoré ho kontrolujú, upravujú a dokonca analyzujú aj počas ich behu, ešte pred tým ako sa skompilujú do *bajtkódu*. To nám otvára svet možností pre introspeciu a testovanie.

Existujú aj prípady kde sa vychádza z *bajtkódu* interpretu [28] [6].

3.2.1 Statická analýza

Úlohou statickej analýzy je skúmanie zdrojového kódu bez jeho spustenia za účelom identifikovania potencionálných chýb, prípadne problémov kvality pred tým ako sa dostane do produkcie. Počas tejto analýzy sa využíva prechádzanie po celom *abstraknom syntaktickom strome*, čo nám umožnuje overenie správnosti programu, nájdenie možnosti zjednodušenia a vylepšenia kódu. V prípade staticky typovaných jazykov ako napr. Java sú tieto typy analyzérov podstane užitočnejšie a funkčnejšie ako v Pythone. Tieto jazyky majú typy premenných a funkcií známe už v čase prekladu a nedovoľujú, prípadne len veľmi obmedzene, zmenu ich kódu počas behu programu. Vďaka tomu má analyzátor oveľa väčšiu istotu o stave toku a type dát v programe. Analýza toku dát je techika používaná na sledovanie ako sa v čase menia premenné a ich hodnoty [4]. Dynamickosť jazyka Python zabraňuje efektívnej statickej analýze, robí ju komplikovanú a limitovanú, čo sa týka množstva detekcie problémov, ktoré je schopná zachytiť [6]. Ak sa pozrieme na ukážku kódu 3.3, tak nie je možné s naprostou istotou povedať aké majú typy premenné `a` a `b`, môže to byť typ `int`, `str` prípadne akákoľvek trieda, ktorá implementuje metódy `__mul__` a `__add__`. Statické analyzéry ako napríklad Pylint využívajú rôzne heuristiky, aby si s týmto problémom pomohli, ale ma to za následok vznik tzv. *false positives* [25].

3.2.2 Dynamická analýza

Dynamická analýza využíva znalosti o programe, ktoré má k dispozícii počas jeho spustenia, a preto je schopná detekovať závislosti a informácie, ktoré nie sú možné zistiť pomocou statickej analýzy. Dobrým príkladom sú závoslosti používajúce reflekciu[5], injektovanie závislostí[6] alebo polymorfizmus. Jedným z možných nevýhod použitia dynamickej analýzy je, že sa môže zhoršiť výkon aplikácie počas testovania.

Príklady dynamickej analýzy:

- *Analýza pokrytia kódu* – Meranie pokrytia kódu, príkazov nebo rozhodovacích konštrukcií
- *Jednotkové, regresné, integračné testovanie* – Spôsob overovania kvality systému založená na vykonávaní daného systému pomocou testov
- *Assertions* – Kusy kódu vytvorené programátorom, ktorých úlohou je pri nesplnení nejakej nutnej podmienky ukončíť program a upozorniť, že k takejto situácii došlo.
- *Black/Gray/White box testovanie* – Testovanie s určitou transparentnosťou systému, ktoré určuje ako testujeme samotný systém

[4] Dataflow analysis: http://www.seas.harvard.edu/courses/cs252/2011sp/slides/Lec02-Dataflow.pdf

[5] Reflection: http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html

[6] Dependency injection: https://msdn.microsoft.com/en-us/library/dn178469(v=pandp.30).aspx

Kapitola 4

Porovnanie existujících riešení

V tejto kapitole sa zameriavame na konkurečné riešenia, kde porovnávame ich vlastnosti a funkcionalitu. Pri ich testovaní sme ale nenašli žiadne, ktoré by ponúkali vyučbu jazyka Python formou Automatic Tutor s automatickou analýzou za účelom poskytnutia lepších postupov z hľadiska výkonnosti a čistoty kódu. Preto si najprv preberieme tutoring systémy, vizualizátory a nakoniec analyzátory a z každého sa pokúsime vybrať jeho najlepšie vlasnosti a spojiť ich do nášho *Python Tutor* systému.

4.1 Tutoring systémy

Systémy v tejto sekcii fungujú vo forme, že študent dostane úlohu, následne ju vyrieši a odovzdá a dostane spätnú väzbu, či jeho riešenie vyhovuje testovacej sade. Nevykonáva sa tu žiadna analýza.

4.1.1 HackerRank

HackerRank je miesto, kde sa programátori z celého sveta stretávajú s cieľom riešiť úlohy v širokej škále oblastí informatiky, ako sú napríklad algoritmy, strojové učenie alebo umelá inteligencia. Dajú sa tu aj vyskúšať rôzne programovacie paradigmaty, ako napríklad funkcionálne programovanie [8]. Jedným z ponúkaných kurzov je aj kurz jazyka Python[1]. V prílohe na obr. A.1 je zobrazené jeho užívateľské rozhranie.

4.1.2 Codecademy

Codecademy je online interaktívna platforma, ktorá ponúka bezplatné kurzy programových jazykov. V súčasnosti ich podporuje 12 vrátane Pythonu. Jeho unikátnosť spočíva v tom, že poskytuje personalizovaný plán učenia s odporúčanými kľúčovými zručnosťami, ktoré by mal študent nadobudnúť po jeho absolvovaní. Študenti majú možnosť sledovať ich pokrok voči tomuto plánu a môžu si pridávať ďalšie kurzy, podľa vlastného uváženia [1]. Väčšina týchto funkcií je ale sprístupnená až po zaplatení mesačného poplatku, a to je jeho hlavná nevýhoda.

V prílohe na obr. A.2 sa nachádza ukážka prostredia *Codecademy*. Celý systém je založený vo forme lekcií, pričom každá z nich má na jej konci kvíz, prípadne nejakú úlohu, ktorú musí študent vyriešiť na odomknutie ďalšej lekcie. Jeho ďalšia výhoda je, že sa úlohy riešia priamo v prehliadači a nezaťažuje študenta s nastavovaním prostredia.

[1]Kurz jazyka Python v HackerRank: https://www.hackerrank.com/domains/python/py-introduction

4.2 Vizualizačné nástroje

Tieto typy nástrojov sú skvelým pomocníkom pre začínajúcich programátorov. Síce je ich jediná funkcionalita grafické zobrazenie prebiehaného kódu, ale dokáže názorne vysvetliť koncepty. Síce nie sú tématom tejto práce, ale ich možný prínos pre systém *Python Tutor* stojí za zmienku, prvý z nich existuje aj ako knihovna, ktorá sa da vložiť do vlastnej webovej stránky. V budúcnosti by sa mohla integrovať do tohto systému a poskytnúť ešte väčšiu efektivitu učenia a názornosť.

4.2.1 Online Python Tutor

Online Python Tutor je webový vizualizačný nástroj pre jazyk Python, ktorý sa stáva populárnym jazykom pre výučbu úvodných kurzov CS. Použitím tohto nástroja môžu učitelia a študenti písať Python priamo vo webovom prehliadači (bez inštalácie doplnkov), krokovať dopredu a dozadu, vidieť stav dátových štruktúr počas spustenia programu a zdieľať tento program aj s vizualizáciou na webe[31]. Najväčšou výhodou tohto nástroja, je že študenti majú názorné grafické zobrazenie každého stavu programu. (viď obrázok A.3)

4.2.2 Code Skulptor

Code Skulptor je webové programovacie prostredie navrhnuté na podporu výučby Pythonu pre MOOC[35]. Využíva nástroj *Online Python Tutor* na grafické zobrazenie a pridáva podporu jednoduchého GUI[2] na okná a grafickú interakciu. (viď obrázok A.4)

4.3 Analyzačné nástroje

4.3.1 Pylint

Pylint je najrozšírenejší syntaktický analyzátor zdrojového kódu na kontrolu chýb jazyka Python. Dodržiava PEP8[3], čo je štandardný programovací štýl pre jazyk Python. *Pylint* je statický analyzátor, čo znamená, že nespúšťa kód. Dynamická forma Pythonu, mu nedovoľuje zistiť typy všetkých premenných v programe.

4.3.2 Pychecker

PyChecker je nástroj na hľadanie chýb v zdrojovom kóde. Z dôvodu dynamickej povahy Pythonu môžu byť niektoré hlásenia nesprávne, avšak falošné varovania by mali byť pomerne zriedkavé. Jedným z rozdielov oproti *Pylint* a *Pyflakes* je to, že on pri analýze importuje každý modul. To znamená že prebieha aj statická a dynamická analýza. Využíva ale minimum informácii, ktoré môže získať spustením programu. Import poskytuje niektoré základné informácie o module a každá funkcia, trieda a metóda je kontrolována kvôli možným problémom [16]. *Pychecker* funguje len s verziami Pythonu 2.0 až 2.7.

4.3.3 Pyflakes

Pyflakes je podobný ako *Pychecker* ale líši sa tým, že na analýzu nepotrebuje importovať moduly na ich kontrolu. Je to bezpečnejšie a rýchlejšie, aj keď nevykonáva toľko kontrol.

[2] Grafické užívateľské rozhranie
[3] PEP8: https://www.python.org/dev/peps/pep-0008/

Na rozdiel od *PyLint*, *Pyflakes* kontroluje iba logické chyby v programoch. To znamená, že nevykonáva žiadnu kontrolu štýlu kódu [17].

Kapitola 5

Použité technológie

V tejto kapitole sa zoznámime s technológiami, ktoré boli potrebné pre vytvorenie tejto bakalárskej práce. Bude tu vysvetlené prečo sú používané zmienené jazyky, technológie alebo knihovny a porovnanie s možnými alternatívami. Znalosť nasledujúcich technológií je kľúčová pre pochopenie procesu vytvárania bakalárskej práce.

5.1 Python

Na zbieranie typov premenných počas behu programu sme potrebovali veľmi interoperabilný prístup k spustenému programu. Prichádzalo jediné riešenie, a to použitie jazyka Python a jeho štandardnej implementácie CPython. Ako sme už spomínali v kapitole 3, python je veľmi dynamický jazyk, ktorý ma dokonca implmentovaný debugger v štandardnej knihovne. Je ho možné upraviť pre vlasné potreby. V práci je využitý Python vo verzii 3.6.1, vďaka čomu sme mohli využiť novinky ako *gradual typing* na zjednodušenie a zefektívnenie vývoja.

5.2 Astroid

Jazyk Python obsahuje v štandardnej knihovne modul `ast`, ktorý dokáže prekonvertovať zdrojový kód na abstraktný syntaktický strom. Pre potreby tejto práce bol príliš jednoduchý, a preto sme zo začiatku vlastnoručne implementovali dodatočnú funkcionalitu. V priebehu práce sme objavili knihovnu Astrioid, ktorá ma rovnaký cieľ ako modul `ast`, ale je využiteľnejšia pre naše potreby.

5.3 XML a XPath

XML dokument tvorí stromovú štruktúru. Za jeho nevýhodu môžeme považovať komplexnosť, ale jeho hlavná výhoda spočíva v množstve informácií, ktoré je schopný uchovať a to pri zachovaní čitateľnosti pre stroje i ľudí. Dalšia výhoda je existencia dotazovacieho jazyka XPath, ktory je špecifikovaný štandardom [26] a ma kvalitnú podporu knihoven [11]. V tejto práci je použitá LXML[1], ktorá je obal nad libxml2[2], čo nám umožňuje dostatočnú rýchlosť pri pracovaní s komplexnými XML súbormi. XPath poskytuje možnosti navigácie v strome, prípadne výber uzlov podľa rôznych kritérií. Podrobnejšie sa mu venuje sekcia 6.1.7.

5.4 Flask

Pre Python existuje mnoho webových frameworkov na server side development. Frameworky typu full-stack ako je napríklad *Django* sú zbytočne komplexné. V tomto projekte by používali veľa zdrojov a pritom by sa využil len zlomok potenciálu celého frameworku. Kedže komunikácia medzi klientom a serverom priebeha cez nenáročné REST[3] API[4], tak sa pre serverovú časť vybral *Flask*. Flask je modulárny webový framework a v prípade potreby je možné stiahnuť balíčky a rozšíriť tým jeho funkcionalitu.

5.5 Docker

Obecne testovanie a spúšťanie kódu z nedôverihodných zdrojov akéhokoľvek jazyka je nebezpečná vec. Tým, že je jazyk Python tak dynamický, to platí obzvlášť, pretože povoľuje meniť kód počas behu programu. Počas behu programu dokáže injektovať proces importovania modulov, prípadne podvrhnúť `builtins` (funkcie a premenné definovnané implicitne v každom programe) a funkcie zo štandardnej knihovny.

V minulosti existoval projekt `pysandbox` [12], ktorý mal zabrániť takýmto možnostiam, ale existuje príliš veľa spôsobov ako uniknúť z izolovaného prostredia nedôverihodnému kódu pomocou rôznych funkcií introspekcie jazkyka Python.

Jedno z odporúčaných a overených riešení je využitie virtualizácie pomocou kontajnerov [27]. V súčasnosti existuje mnoho riešení, každé majú svoje výhody a nevýhody. Jedno z najjednoduchších riešení na manažment je Docker, ktorého cieľom je poskytnúť jednotné rozhranie a izoláciu pre aplikácie. Poskytuje ďalšiu vrstvu abstrakcie a automatizácie na úrovni operačného systému. Využíva vlastnosti izolácie zdrojov jadra Linuxu, ako sú `cgroups` a `kernel namespaces` a súborový systém `OverlayFS`, ktoré mu umožňujú spúštanie na sebe nezávislých kontajnerov na hostovskom operačnom systéme. Réžia na spustenie a manažment kontajnerov je vďaka tomu oproti konkurenčným riešeniam ako Virtualbox minimálna.

[1] Python knihovna na pracu s XML: http://lxml.de/
[2] The XML C parser and toolkit of Gnome: http://xmlsoft.org/
[3] Representational state transfer: https://www.w3.org/2001/sw/wiki/REST
[4] Application Programming Interface

5.6 AngularJS

AngularJS je javascriptový MVC (Model View Controller) framework vytvorený spoločnosťou Google na správnu tvorbu architektúry za účelom udržiavateľnosti webových aplikácií. AngularJS je postavený na filozofii, že deklaratívny kód je lepší než imperatívny, pri budovaní UI[5] sa využívajú komponenty [32].

Klientská časť aplikácie je rozdelená na komponenty, pričom každý z nich má zodpovednosť za špecifickú funcionalitu. Ich spojením, pomocou jasne definovaných rozhraní, sme docielili do budúcnosti rozšíriteľný a udržiateľný kód.

5.7 Code Mirror

CodeMirror je textový editor spustiteľný v prehliadači [3]. Je použitý v *read-only* režime na interaktívne zobrazovanie výsledku analýzy.

[5] User Interface

Kapitola 6

Návrh riešenia a implementácia

Po výskume aktuálnych riešení sme sa rozhodli implementovať systém s architektúrou klient/server. Medzi hlavné výhody tohto spôsobu realizácie patria:

- *žiadna konfigurácia a nastavovanie* – Jedna z najdôležitejších vecí pre začínajúcich študentov. Docielime tým zníženie zaťaženie študentov na minimum.
- *aktuálnosť* – V prípade skriptu spustiteľného na lokálnom stroji by bolo omnoho ťažšie, ba priamo nemožné, zosynchronizovať najnovšiu verziu Tutor systému väčšiemu počtu študentov bez nejakého automatického upgradovacieho mechanizmu. Pri tomto type architektúry nám tento problém odpadá a všetci študenti budú mať vždy najaktuálnejšiu verziu programu.

Ako sme v sekcii 3.2 naznačili výhody a nevýhody statickej a dynamickej analýzy, tak sme sa rozhodli využiť výhody ich oboch. Počas behu programu nazbierame informácie (typy premenných), ktoré následne vložíme do abstraktného syntaktického programu. Tým docielime to, že napriek tomu že je Python dynamický jazyk, bez deklarovaných typov v čase písania kódu, môžeme ich mať k dispozíci počas statickej analýzy a pracovať s nimi, ako keby ich mal definované.

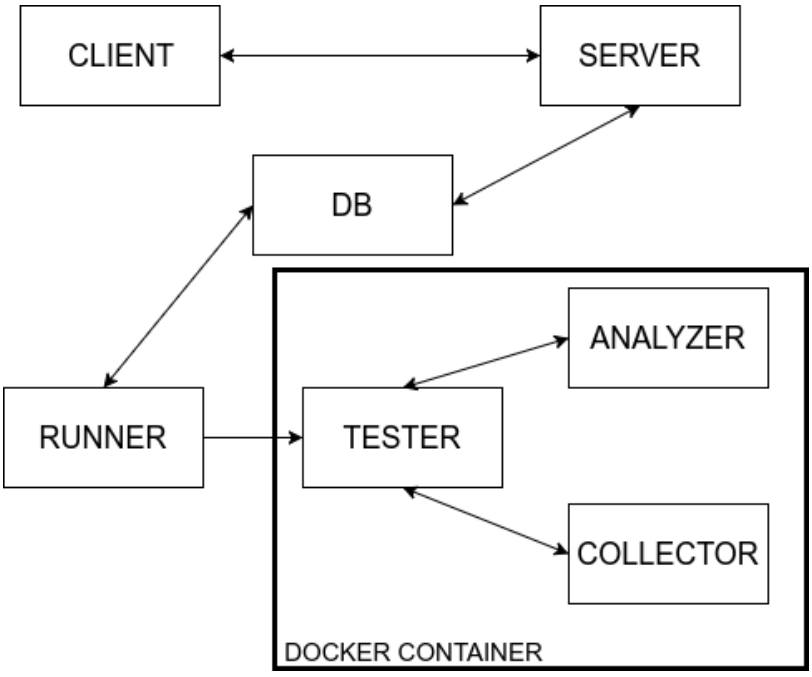
Pre potreby tejto práce boli vyvinuté 2 analyzátory, ktoré obe vychádzajú z abstraktného syntaktického stromu:

- *Dynamic analyzer* obohatený o statické typové informácie z behu programu
- *XPath analyzer* vužívajúci XML formát, ktorý je dotazován technológiou XPath

Problematikou analýzy sa podrobnejšie zaoberá sekcia 6.1.7.

6.1 Moduly systému

Pre jednoduchší pochopenie a debuggovateľnosť bola navrhnutá modulárna štrukúra. Ďalším dôvodom je schopnosť vyextrahovať functionalitu, ktorú by bolo možné použiť v inom projekte. Architektúra systému *Python Tutor* je zobrazená na obrázku 6.1.



Obr. 6.1: Architektúra Automatic Tutor Python systému

6.1.1 Client

V tejto časti sa budeme venovať modulu *Client*. Jedná sa o prvky aplikácie, ktoré sú viditeľné pre užívateľa.

Jeho úlohou je interakcia užívateľa s modulom *Server*. Ako je možné si všimnúť na obrázku 6.3 využívame *Material design*[1] kvôli jeho prehľadnosti a jednoduchosti.

Client poskytuje 2 užívateľské módy – študent, učiteľ. Využívajú sa rovnaké komponenty pre oboch s rozdielom, že študent nemá niektoré funkcie sprístupnené.

Zobrazenie úloh na riešenie (obr. 6.3)

V ľavej hornej časti navigačnej lišty sa nachádza tlačidlo *TASKS*. Po jeho kliknutí sa nám zobrazia úlohy na riešenie. Farebný pás určuje jeho obtiažnosť. *Červená* značí najviac obtiažne, *zelená* naopak jednoduché a *oranžová* je stredná obtiažnosť.

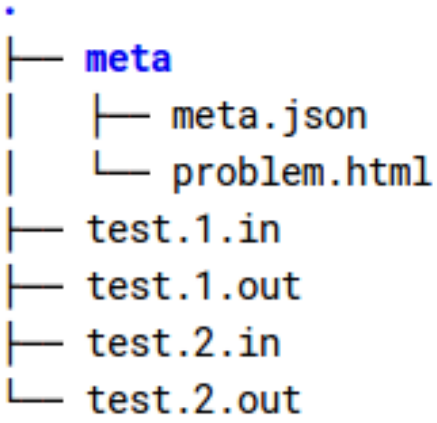
Učiteľ ma možnosť vymazávať úlohy, prípade vložiť novú.

Vytvorenie novej úlohy

Zatiaľ, čo zobrazenie úloh na riešenie je pre študenta aj učiteľa rovnaké, vytvorenie novej úlohy, alebo jej odstránenie, prináleží len učiteľovi.

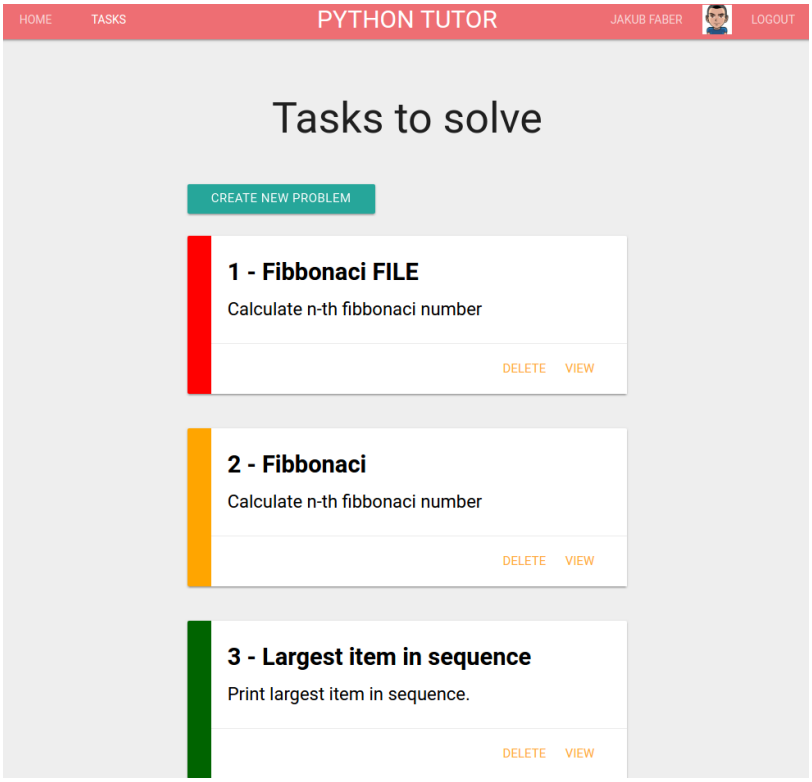
[1] Material Design CSS knihovna: http://materializecss.com/

Učiteľ má možnosť v prehľade aktuálnych úloh (obr. 6.3) vytvoriť novú tlačítkom *CREATE NEW PROBLEM*. Kliknutím sa vyvolá okno pre nahrávanie nových úloh spolu s inštrukciami a pokynmi, čo ma archív obsahovať (obr. 6.2). V prílohe B je detailnejšie vysvetlenie, čo musí byť obsahom jednotlivých súborov.



Obr. 6.2: Povinný obsah zip súboru

Ak by sme sa pokúsili nahrať formát iný ako *zip* alebo ak štruktúra súborov v archíve nezodpovedá pokynom, tak nás system upozorní notifikáciou.

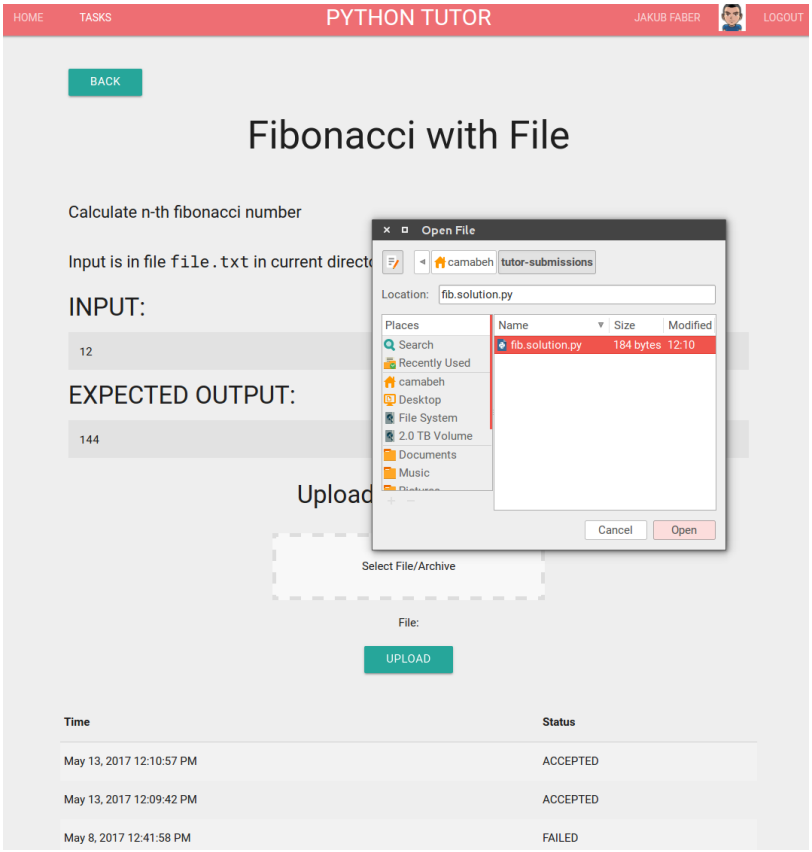


Obr. 6.3: *Python Tutor*: Zobrazenie úloh na riešenie

Zobrazenie úlohy a nahranie jej riešenia (obr. 6.4)

Bez prihlásenia do systému je povolené len zobrazovanie úloh, nemôžu vkladať svoje riešenia. Registrácia je rychlá a bezplatná, stačí sa prihlásiť pomocou Google účtu. Na *Serveri* sa uchovávajú len základné údaje ako je emailová adresa, meno, priezvisko, odkaz na profiloú ikonu užívateľa a token.

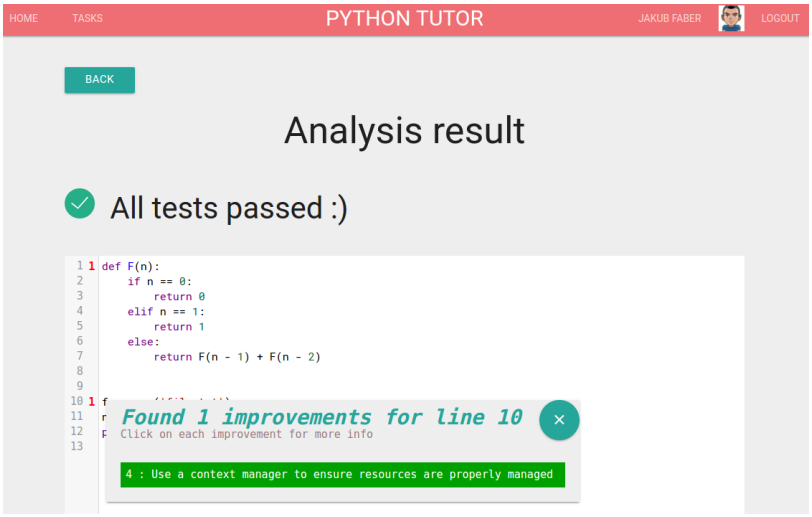
Po tom čo užívateľ nahraje riešenie na server, dôjde k spusteniu procesu testovania a analýzy. O jeho priebehu spracovania ho informuje tabuľka v dolnej časti a notifikácia.



Obr. 6.4: *Python Tutor*: Zobrazenie úlohy a odovzdávanie riešenia

Interaktívny editor s výsledkom testov a analýzy (obr. 6.5)

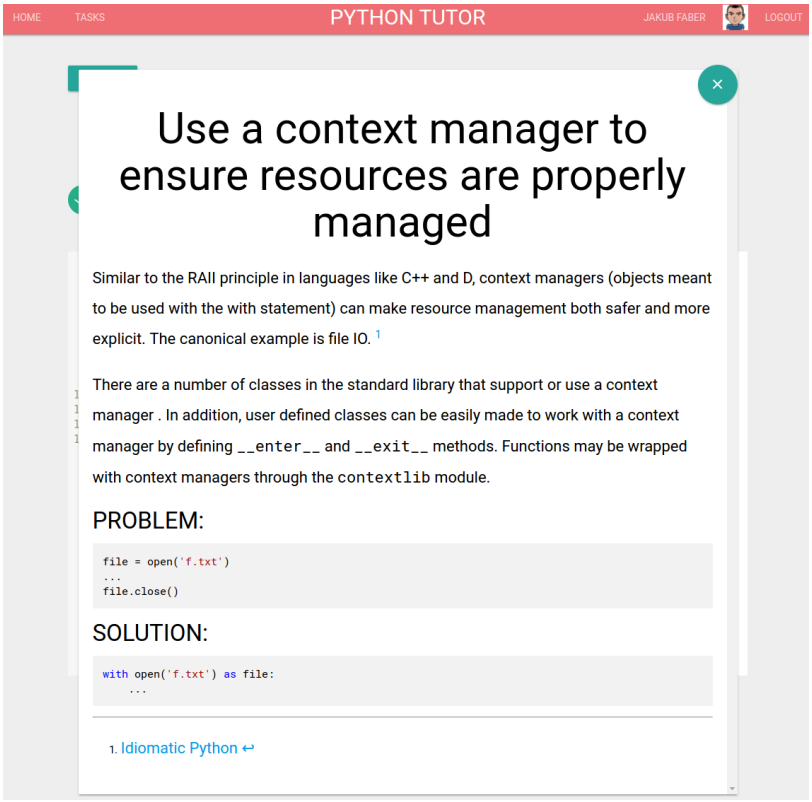
Po prijatí výsledku testovania si užívateľ môže kliknutím na status alebo dátum odovzdaného riešenia otvoriť okno s analýzou, v ktorom sa mu zobrazí detailnejší výstup v prípade testu, ktorý neprešiel spolu s interaktívnym editorom. Na pravo od čísla riadku je zobrazený počet *improvements*, ktoré *Analyzer* navrhol. Po kliknutí na ne sa otvorí tabuľka, kde sú všetky podrobnejšie vypísané.



Obr. 6.5: *Python Tutor*: Interaktívny editor s výsledkom testov a analýzy

Návrh na zlepšenie kódu s popisom a príkladom (obr. 6.6)

V prípade, že sa chce užívateľ dozvedieť o konkrétnej možnosti vylepšenia, ktoré doporučuje *Analyzer*, tak musí kliknúť na jedno z nich a otvorí sa mu okno s podrobnejším popisom. Každá *improvement* vždy obsahuje nevhodný, prípadne neidiomatický kód a alternatívu k nemu. V dolnej časti sa nachádza odkaz na zdroj z ktorého boli čerpané informácie.



Obr. 6.6: *Python Tutor*: Návrh na zlepšenie kódu s popisom a príkladom

6.1.2 Server

Flask

Využívame framework Flask pre jeho jednoduchosť. Pre dodatočnú funkcionalitu ako je prihlasovanie užívateľov používame knihovnu `flask_login`[2]. Aplikácie je napísaná spôsobom, aby šla bez problémov spravovať a rozširovať.

Šifrovanie

Server používa šifrovanie *TLS 1.2* na zabezpečenie komunikácie medzi modulmi *Client* a *Server*. Vlastnoručne podpísaný certifikát je vygenerovaný programom `openssl`[3]. V súčasnosti prehliadače hlásia problém, že webový server nie je dôverihodný, pretože jeho certifikát nepodpísala vierohodná certifikačná autorita. V ostrej prevádzke by sa využila certifikačná autorita *Let's Encrypt* a tým by bol tento problém vyriešený [4].

Rozhranie REST API

Na klientovi používame *AngularJS* ako framework za účelom dosiahnutia SPA[5]. Pre neho je typické, že sa používa REST rozhranie na komunikáciu. Jeden z dôvodov je, odľahčenie serveru od neustáleho generovania dynamických šablón. Dalším, pravdepodobne dôležitejším, je že technológia REST nie je závislá na operačnom systéme ani platforme. Vďaka tomu je možné vytvoriť aplikáciu pre každý desktopový aj mobilný operačný systém, a pritom všetky môžu využívať rovnaký server. Je to obrovská úspora času, ktorý sa dá využiť napríklad na tvorbu novej funkcionality. Na serializáciu Python objektov do JSON a opačne využívame knihovnu `marshmallow`[6].

6.1.3 Databáza

Na vývoj bola použitá databáza SQLite vo verzii 3. Jej výhodou je, že nemusí bežať v samostatnom procese a umožňuje nám jednoduchý prístup pomocou SQL. Často sa využíva pri prototypovaní, ale nie je problém časom migrovať dáta do väčšej databázy, ako je napríklad PostgreSQL alebo Oracle. Aby sme nemuseli pri zmene databáze prepisovať SQL dotazy, tak využívame knihovnu SQLAlchemy[7], ktorá poskytuje deklaratívne API a transparentne generuje dotazy SQL podľa typu pripojenej databáze.

6.1.4 Runner

V tejto sekcii sa budeme venovať modulu, ktorý je zodpovedný za spúšťanie testov.

Ako sme už naznačili v sekcii Docker (viz. 5.5) v kapitole o použitých technológiach, spúšťanie programov v izolovanom prostredí nie je vôbec jednoduché. Práve preto používame riešenie *Docker* na bezpečné odizolovanie skriptov študentov v kontajneri, ktorý nemá prístup ani na hosťovský systém ani na sieť. Využíva sa tzv. *Docker image*, čo je predpripravené prostredie pripravené autorom práce. V prílohe **??** sa nachádza zoznam súborov, ktoré

[2] Flask_login: https://flask-login.readthedocs.io/en/latest/
[3] OpenSSL: https://www.openssl.org/
[4] Certifikačná autorita Let's Encrypt: https://letsencrypt.org/
[5] Single Page Application
[6] Mashmallow serializačná knihovna: https://marshmallow.readthedocs.io/en/latest/
[7] SQLAlchemy: https://www.sqlalchemy.org/

sú používané a ako sú namapované na hosťovský operačný systém a postup ako vygenerovať *Docker image* na novom systéme.

Runner sa spúšťa vo vlastnom procese, oddelene od modulu *Server*. Je to z dôvodu, že webový framework Flask je blocking – to znamená, že dlho bežiace úlohy ako je napríklad spúšťanie testov a analýza by úplne zablokovali server a pre viacerých užívateľov by sa stal systém nepoužiteľný.

Runnner je napojený na rovnakú databázu ako *Server* a každých 15 sekúnd si vytiahne záznamy z tabuľky `submission`, ktoré ešte nie sú zanalyzované a otestované. V prípade nálezu sa spustí predpripravený *Docker image* spolu s namapovanými cestami k testovanému skriptu a úlohe. Ak sa skript zacyklí, alebo nestihne ukončiť v určitom časovom intervale, tak je tento kontajner zrušený a skript je považovaný za nefunkčný.

Ak sa žiaden záznam nenájde, ostáva v nečinnosti a po uplynutí 15-tich sekúnd sa to opakuje.

6.1.5 Tester

Tester modul je spúšťaný v izolovanom prostredí kontajneru *Docker*. Jeho vstupum sú 2 namapované priečinky.

- `/problem` obsahuje úlohu, ktorej sa test týka, ktorá ma štruktúru ako na obr. 6.2
- `/submission` obsahuje študentom odovzdaný súbor `submission.py`

Podľa metadát zo súboru `/problem/meta/meta.json` *Tester* nastaví buď súbor alebo štandardný vstup ako vstup pre testovaný skript. Testy musia byt pomenované `test.N.[in|out]` kde `N` je > 0. Po nastavení vstupu sa zavolá metóda `perform_first_test`, ktorá spustí prvý test pod dohľadom *Collector* (viď sekcia 6.1.6). Po úspešnom dokončení (program neskončil s výnimkou alebo na timeout) sa spustí analýza metódou `analyze` triedy `Analyzer` z modulu *Analyzer* (viď sekcia 6.1.7).

Výsledkom analýzy je súbor `analysis_result.json` v priečinku `/submission` vo formáte *JSON*[8].

Na koniec prebehnú ostatné testy a výsledky zo všetkých testov sa uložia ako `test_result.json` v priečinku `/submission`.

Oba súbory si prevezme *Runner* a uloží ich obsah do tabuľky `submission` do databáze.

6.1.6 Collector

Táto sekcia popisuje modul na zbieranie informácií a typov premenných z behu programu, ktoré sa následne využívaju v *Analyzer*.

Štandardná knihovna jazyka Python obsahuje debugger, ktorý sa dá spustiť jednoducho príkazom `python3 -m pdb myscript.py`[9] z príkazového riadku alebo kdekoľvek do kódu vložiť `import pdb; pdb.set_trace()` a pri prevádzaní tohto riadku sa spustí debugger s aktuálnym kontextom.

Pre potreby tejto práce debugger z modulu `pdb` pracuje na príliš vysokej úrovni, obsahuje totiž vlasnú logiku na interakciu s užívateľom. My by sme však potrebovali pracovať s debuggerom programaticky, vložiť do neho vlasnú logiku. V knihovne existuje modul `bdb`[10], ktorý je používaný na implementáciu alternatývych debuggerov [10].

[8] Javascript Object Notation Object - serializačný formát
[9] `pdb` modul: https://docs.python.org/3/library/pdb.html
[10] `bdb` modul: https://docs.python.org/3.6/library/bdb.html

Trieda `Collector` dedí funkcionalitu z `bdb.Bdb`, čo nám umožňuje zasahovať do programu počas behu. Pre umožnenie interakcie musíme preťažiť niektoré z metód `user_call`, `user_line`, `user_exception` alebo `user_return`. Nám stačí tá posledná, pretože pri každom navrátení z akejkoľvek funkcie, ale ešte pred navrátením sa k miestu volania, program skočí do `user_return`. To nám dáva možnosť zistiť typy premenných, ktoré vznikli v tejto funkcii. Všetky takto vytvorené premenné patria do menného priestoru funkcie [18]. Pre spustenie skriptu pod kontrolou triedy `Collector` sa musí zavolať metóda `collect_types`, ktorá po jeho úspešnom dokončení vráti objekt typu `RuntimeMetadata`. Ten obsahuje metadata o typoch premenných z behu programu. Tie sa využívajú v *Analyzer* module.

Jedna z nevýhod tohto prístupu, je že získame typy len v častiach kódu, ktoré boli spustené. V zvyšných častiach, ktoré neprebehli nemáme žiadne informácie a sme na tom, ako keby sme ho nepoužili. Každopádne, to nie je prekážka, lebo máme aspoň informácie o typoch v časti ktorá prebehla.

6.1.7 Analyzer

Dostávame sa k jádru *Python Tutor* systému. Opíšeme si aké metódy analýzy kódu využíva, aké sú možnosti rozšírenia novými *inspections* a ako využíva runtime metadata nazbierané v module *Collector*. Chcel by som upozorniť na možné zamenenie terminológie, kým v opise *Client* a *Server* sme hovorili o výsledku analýzy ako *improvement*. V tejto časti to budeme nazývať *inspection*, pretože v tomto kontexte to nie výsledok, ale zdrojový popis chyby, ktorá sa má hľadať v zdrojovom kóde študenta. Súbor `analyzer_.py` obsahuje triedu `Analyzer`, ktorá má za úlohu pomocou oboch analyzátorov (popísaných nižšie) objaviť možné vylepšenia v testovanom programe.

Dynamický Analyzer obohatený o runtime metadata

Ako už bolo naznačené na začiatku kapitoly 6, tak pre túto bakalársku prácu je statický syntaktický analyzátor obohatený o metadata, ktoré obsahujú typy premenných z čase behu programu. Samozrejme, že to nie je 100% bezchybné, pretože ako ukazuje kód v ukážke 6.1, v priebehu funkcie `fun1` sa vystriedali 3 typy pre premennú `a`. V prvom prípade je to `int`, `str` a nakoniec `builtin_function_or_method`. V sekcii 6.1.6 bola vysvetlená aká technika sa používa na extrahovanie typov z bežiaceho kódu. V súčasnosti nám bohužiaľ nedovoľuje zachytiť na každom riadku kódu správny typ, pretože až po ukončení funkcie budú k dispozícii typy pre premnné `k`, `b` a `a`, pričom typ `b` závisí od toho aký bude typ vstupného parametra `k`.

```
def fun2(i):
    return i * 5

def rectangle_circ(a, b):
    return 2 * (a + b)

def fun1(k):
    b = list(map(fun2, k))
    a = 8
    a = "Hello world"
    a = min
```

Ukážka 6.1: Príklad dynamickosti jazyka Python

Konštruktor objektu `Ast` potrebuje 2 parametry, aby dokázal vytvoriť abstraktný syntaktický strom s runtime metadátami. Tým prvým je testovaný skript reprezentovaný ako reťazec znakov, a tým druhým je objekt typu `RuntimeMetadata`, ktorý sme získali z modulu *Collector*.

Následne sa pomocou knihovny `astroid` vygeneruje klasický abstraktný syntaktický strom (viď obr. 6.7). Po jeho vytvorení sa musia navštíviť všetky uzly, ktoré majú typ funkcie, pretože do nich chceme vložiť informácie o typoch, ktoré sme získali. Objekt `RuntimeMetadata` obsahuje slovník (dictionary), kde jeho kľúčom je reťazec `name:lineno`, pričom `name` je názov funkcie a `lineno` je riadok, na ktorom je táto funkcia definovaná. Pod týmto kľúčom sa v ňom nachádza objekt `RuntimeFunctionMetadata`, ktorý je používaný na uloženie typov lokálných premenných danej funkcie. Mohla by nastať situácia, že istá časť kódu (funkcia) nebola prevedená, a tak nebude mať pridelený `RuntimeFunctionMetadata` objekt. V takomto prípade bude vykonaná analýza v tejto časti kódu bez znalosti skutočných typov.

```
try:
    os.remove('somefile.tmp')
except FileNotFoundError:
    pass
```

Ukážka 6.2: Nevhodné použitie

```
with suppress(
    FileNotFoundError):
     os.remove('somefile.tmp')
```

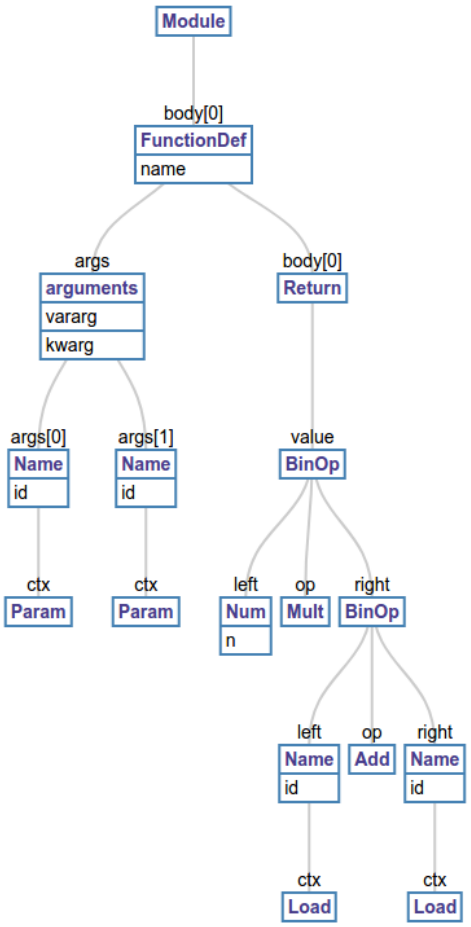
Ukážka 6.3: Využite context manageru `suppress` z modulu `contextlib`

Pridávanie nových *inspections*: Uvedieme si to na príklade. Predstavme si, že chceme pridať *inspection* z nevhodného kódu v ukážke 6.2 na vhodný v ukážke 6.3. Kroky ktoré musíme vykonať:

- pridať element do Enum triedy v súbore `analyzer/constants.py` a prideliť mu unikátne **ID**
- vytvoriť visitor – Funkciu, ktorá bude automaticky volaná pri prechádzaní abstraktného syntaktického stromu. V prípade že sa narazí na uzel, ktorého typ sa v našom

prípade zhoduje s `astroid.nodes.TryExcept`, tak vykonať analýzu tejto časti kódu a následne v prípade chyby vrátiť Enum typ z predchádzajúceho kroku. (viď. súbor `analyzer/inspections/try_.py`)

- zaregistrovať tento visitor do triedy `InspectionManager`, aby bol volaný automaticky
- vytvoriť **ID**.md (**ID** z prvého kroku) a uložiť ho do `resources/inspections/`. Tento markdown súbor musí obsahovať aspoň jeden element typu `#` (pri spustení serveru sa všetky `*.md` súbory prekonvertujú na `*.html` a z nich sa vyextrahuje hodnota html elementu `h1` a tá sa uloží do databáze ako popisek tejto `inspection`)



Obr. 6.7: Zjednodušený AST funkcie `rectangle_circ` z ukážky 6.1 pomocou [19]

Obr. 6.8: AST reprezentovaný ako XML z funkcie `rectangle_circ` z ukážky 6.1

XPath analyzer

Druhým typom analyzátoru použitého v práci je syntaktický analyzátor založený na vyhľadávaní v abstraktnom syntaktickom strome reprezentovaným formátom XML. Funkcia `ast_to_xml` prevedie tento strom do XML súboru (viď obr. 6.7). To nám dovoľuje dotazovať sa na neho pomocou XPath.

Hlavný rozdiel medzi *Dynamickým analyzátorom* je v tom, že tu nevyužívame runtime metadata. Taktiež nám neposkytuje až také možnosti, ale výroba jednoduchších *inspections* je v ňom menej náročná.

Príklad: XPath výraz: `//body/*[1][not(//Call/func/Name[@id="max"])]`. Ak sa v zdrojovom kóde nenachádza volanie funcie `max`, tak vráti prvý prvok v syntaktickom strome nachádzajúcim sa v tele hlavného modulu.

Pridávanie nových *inspections*: Je jednoduchá, stačí len splniť tieto 3 kroky:

- pridať element do Enum triedy v súbore `analyzer/constants.py` a prideliť mu unikátne **ID**
- vytvoriť výraz v súbore `analyzer/xpath_resource.py` a doplniť mu Enum typ vytvorený v predchádzajucom kroku
- vytvoriť **ID**.md (**ID** z prvého kroku) a uložiť ho do `resources/inspections/`. Tento markdown súbor musí obsahovať aspoň jeden element typu `#` (pri spustení serveru sa všetky `*.md` súbory prekonvertujú na `*.html` a z nich sa vyextrahuje hodnota html elementu `h1` a tá sa uloží do databáze ako popisek tejto `inspection`)

Generovanie XML z ľubovolného zdrojového kódu spustíme príkazom `python3 analyzer/xml_analyzer.py NAZOV_SUBORU.py`. Na debuggovanie XPath existuje vynikajúci online nástroj XPathTool[11].

Príklad: XPath výraz `/Module/body/ImportFrom[names/alias[@name='*']]` nájde všetky riadky v súbore ktoré obsahujú jazykovú konštrukciu `from ... import *`

[11] http://www.qutoric.com/xslt/analyser/xpathtool.html

Kapitola 7

Testovanie

Testovanie funkcionality analýzatorov prebiehalo na sade úloh z predmetu Skriptovacie jazyky z roku 2016, ktoré boli poskytnuté vedúcim práce. Modulárnosť systému *Python Tutor* nám umožnila využiť modul *Analyzer* oddelene a využiť jeho funkcionalitu aj bez nutnosti použitia webového rozhrania.

V tejto sekcii si stručne predstavíme tieto úlohy, a pri každej ukážeme najčastejšie chyby študentov, ktoré objavil. Boli otestované len riešenia, ktoré šli spustiť a nespadli počas behu. Netestujeme správnosť riešenia študentov, ale funkčnosť analyzéru.

V legende nájdeme stručný popis chýb získaných analýzou. Kvôli možnému medzinárodnému využitiu sa v systéme používa anglický jazyk.

Zdrojové podklady, z ktorých čerpá táto kapitola nájdeme v priečinku `data` v koreňovom adresári projektu. V podpriečinku `students_data` nájdeme vstupné súbory pre tieto úlohy a riešenia študentov. V priečinku `data/students_data/proj1/out`, `data/students_data/proj2/out` a `data/students_data/proj3/out` nájdeme výstupy z modulu *Analyzer*. Pre ich opätovné vygenerovanie môžeme spustiť shellový skript `perform_analysis.sh`. Po dokončení nájdeme výsledky analýzy v súboroch s príponou `json` v podpriečinku `out` každej úlohy.

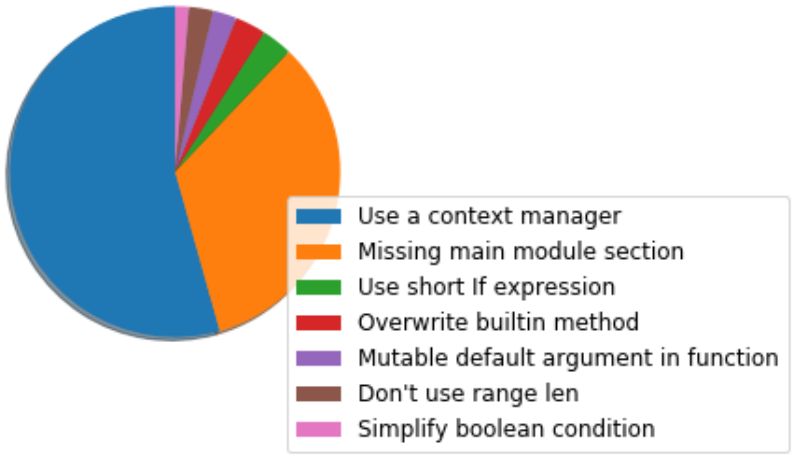
Grafy, ktoré sú uvedené v tejto kapitole boli generované v prostredí *ipython notebook* súborom `make_pie_charts.ipynb`.

7.1 Analýza úlohy č. 1

Prvá úloha bola zameraná na využite regulárnych výrazov.

Na riadkoch, ktore obsahujú reťazec `xkcd` nahradiť text odpovedajúci regulárnemu výrazu `bu.*ls` reťazcom `[gikuj]..n|a.[alt]|[pivo].l|i..o|[jocy]e|sh|di|oo`. Ďalej na riadkoch obsahujúcich znaky `=` a `[` a súčasne neobsahujúce znak `!` nahradia výskyty reťazce `xkcd` za užívateľské meno študenta.

Ako vstupný súbor bol použitý `xkcd1313.ipynb`.



Obr. 7.1: Výsledok analýzy projeku č. 1 (111 súborov)

Pri analýze prvej úlohy bolo použitých 111 vstupných riešení študentov. `Analyzer` objavil celkovo 301 možných vylepšení. To vychádza 2.7 chyby na riešenie. Ako si môžeme všimnúť z grafu, tak načastejší typ chyby sa interne nazýva *Use a context manager*. V súbore `resources/inspections/4.md` nájdeme podrobný popis chyby, ale pre názornosť si v ukážke 7.2 a 7.2 ukážeme dotyčný kód. Najviac chýb nájdených v jedom riešení bolo 6.

```
file = open('f.txt')
...
file.close()
```

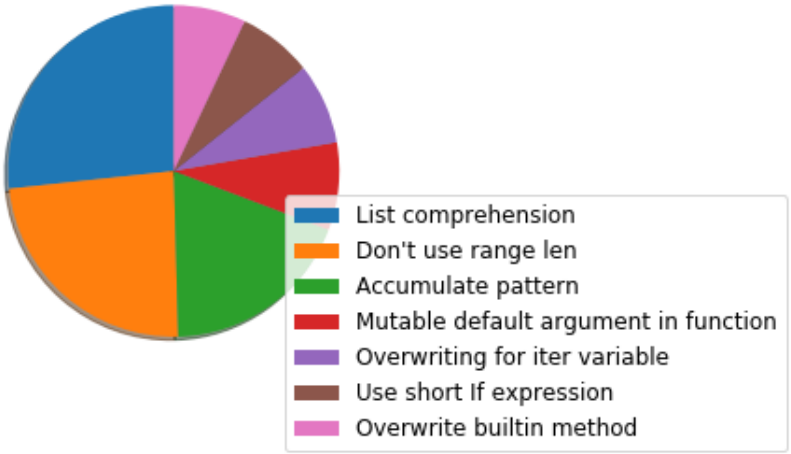
Ukážka 7.1: Chyba `Use a context manager`

```
with open('f.txt') as file:
    ...
```

Ukážka 7.2: Vylepšenie `Use a context manager`

7.2 Analýza úlohy č. 2

Zadaním tejto úlohy bolo implementovať triedu `Polynomial`, ktorá bude pracovať s polynomami reprezentovanými ako zoznamy. Napríklad `2x3 - 3x + 1` bude reprezentované ako `[1, -3, 0, 2]`. Zoznam začína najnižším rádom. Polynomy bude možné sčitať a umocňovať nezápornými číslami a taktiež obsahovať metódy `derivate()` – derivácia a `at_value()` – hodnota polynomu pre zadané `x`.



Obr. 7.2: Výsledok analýzy projeku č. 2 (58 súborov)

Na analýzu bolo použitých 58 súborov od študentov a priemerná chybovosť bola 9.5 na jedno riešenie. Rekordmanom je riešenie s 20 chybami nájdenými *Analyzerom*. V tomto prípade bola načastejšia chyba `List comprehention`. V súbore `26.md` v priečinku `resources/inspections/26.md` nájdeme podrobný popis chyby, ale pre názornosť si v ukážke 7.4 a 7.4 ukážeme kód.

```
s = []
for i in [1, 2, 3, 4, 5]:
    s.append(i)
```

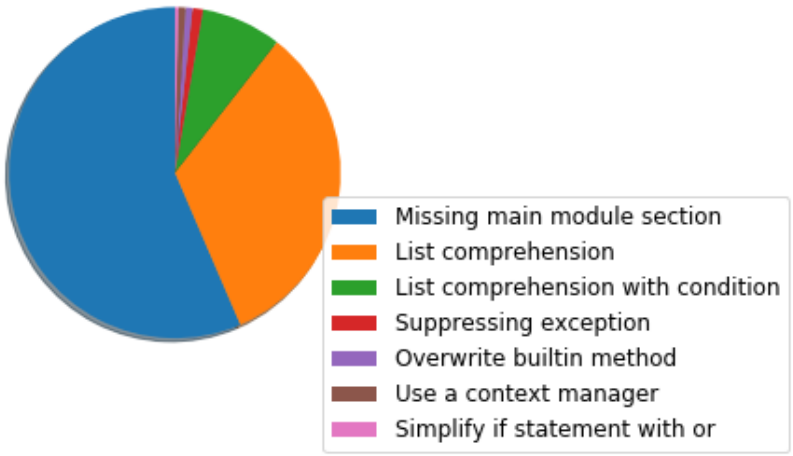
Ukážka 7.3: Chyba `Use a context manager`

```
s = [i for i in [1, 2, 3, 4, 5]]
```

Ukážka 7.4: Vylepšenie `Use a context manager`

7.3 Analýza úlohy č. 3

Cieľom poslednej úlohy bolo naučenie sa pracovať s modulom cProfile, čo je program na profilovanie Python kódu. Praktická časť pozostávala z upravenia zadaného skriptu, aby bežal čo najkratšie a aby zvládal aj extrémne veľké vstupy.



Obr. 7.3: Výsledok analýzy projeku č. 3 (166 súborov)

Analýza prebehla na 166 súboroch. *Analyzer* objavil priemerne 1.8 chyby na jedno riešenie. Najčastejšou chybou je *Missing main module section*. Ukážka 7.5 a 7.6 ukazuje spôsob nevhodného a vhodného požitia. V súbore `15.md` v priečinku `resources/inspections/` je detailné vysvetlenie s výhodami tohto zápisu.

```
...
print('A')
...
```

Ukážka 7.5: Chyba *Missing main module section*

```
def main():
    ...
    print('A')
    ...

if __name__ == '__main__':
    main()
```

Ukážka 7.6: Vylepšenie *Missing main module section*

Kapitola 8

Záver

Po preskúmaní existujúcich možností tutoringu jazyka Python, ktoré neposkytovali dostatočnú spätnú väzbu na kvalitu a čistotu kódu, sa došlo k záveru, že súčasné možnosti sú nedostatočné. S ohľadom na túto skutočnosť bola vytvorená táto práca.

Jej cieľom bolo spojenie automatického tutoring systému *Python Tutor*, v ktorom sa využíva statická a dynamická analýza zdrojového kódu. Pre pochopenie tejto problematiky boli vysvetlené základné princípy analýzy kódu.

Následne bola overená jeho funcionalita na 3 rôznych úlohách, celkom bolo zanalyzovaných 335 riešení študentov.

Analyzer úspešne všetky zhodnotil a navrhol 1146 možných vylepšení. Druhy chýb sa vrámci naších testovaných úloh líšia v závislosti na typu zadania. Z tohto dôvodu nejde vybrať najčastejšiu chybu študentov.

Počas doby práce na tejto bakalárskej práci vyšiel Python vo verzii 3.6, ktorý pridáva podporu statických typov využitím novej syntaxe[1]. Vďaka modulárnosti programu by bolo možné oddeliť modul *Collector* do oddelenej knihovny. Jej vstupom by bol zdrojový kód, ktorý by sa spustil a počas behu programu by zistil typy premenných, parametrov funkcie alebo návratových hodnôt funkcie. Výstupom knihovny by bol vstupný zdrojový kód obohatený o typy. Deklarovanie typov v kóde nám prináša zefektívnenie vývoja a odchytenie niektorých chýb už v čase písania kódu, nie až pri spustení ako je tomu doteraz. Integrované vývojové prostredia, ako napríklad PyCharm[2], ich už podporujú a vďaka tomu môžu našeptávať s oveľa väčšiou presnosťou ako je tomu doteraz.

Python Tutor je ľahko rozšíriteľný a poskytuje možnosti vlastných úprav a vylepšení. Je ho možné upraviť pre potreby online kurzov, a vďaka tomu poskytnúť spätnej väzby pre študentov.

Pre sprístupnenie systému verejnosti by bolo potrebné pridať ďalšie úlohy a vizuálne zatraktívniť uživateľské rozhranie.

[1] Python so statickými typmi: https://docs.python.org/3/library/typing.html
[2] Integrované užívateľské prostredie PyCharm: https://www.jetbrains.com/pycharm/

Literatúra

[1] *Codecademy*. [Online; navštíveno 24.4.2017].
URL https://www.codecademy.com/learn/python

[2] *Codecademy, kurz jazyka Python*. [Online; navštíveno 24.4.2017].
URL https://www.codecademy.com/learn/python

[3] *Code Mirror Editor*. [Online; navštíveno 23.4.2017].
URL https://codemirror.net

[4] *CodeSkulptor*. [Online; navštíveno 24.4.2017].
URL http://www.codeskulptor.org/

[5] *COMPILERS AND TRANSLATORS*. [Online; navštíveno 27.4.2017].
URL https://courses.cs.vt.edu/~cs1104/Compilers/Compilers.070.html

[6] *Evaluating the dynamic behaviour of Python applications*. [Online; navštíveno 15.4.2017].
URL http://crpit.scem.westernsydney.edu.au/confpapers/CRPITV91Holkner.pdf

[7] *Exploring Python Code Objects « late.am documentationn*. [Online; navštíveno 13.4.2017].
URL https://late.am/post/2012/03/26/exploring-python-code-objects.html

[8] *HackerRank*. [Online; navštíveno 18.4.2017].
URL https://www.hackerrank.com/faq

[9] *HackerRank*. [Online; navštíveno 24.4.2017].
URL https://www.hackerrank.com/

[10] *Inside the debugger – interview with Elizaveta Shashkova*. [Online; navštíveno 27.4.2017].
URL https://blog.jetbrains.com/pycharm/2017/03/inside-the-debugger-interview-with-elizaveta-shashkova/

[11] *JSON vs XML*. [Online; navštíveno 25.07.2016].
URL http://www.yegor256.com/2015/11/16/json-vs-xml.html

[12] *Knihovna Pybox na izolovanie python programov*. [Online; navštíveno 21.4.2017].
URL https://github.com/haypo/pysandbox

[13] *Massive online courses enrollment*. [Online; navštíveno 24.4.2017].
URL https://techcrunch.com/2014/03/03/study-massive-online-courses-enroll-an-average-of-43000-students-10-completion/

[14] *MOOC enrolment surpassed 35 million in 2015 - ICEF Monitor - Market intelligence for international student recruitment.* [Online; navštíveno 17.4.2017].
URL http://monitor.icef.com/2016/01/mooc-enrolment-surpassed-35-million-in-2015/

[15] *Programming languages ranked by expressiveness.* [Online; navštíveno 19.4.2017].
URL http://redmonk.com/dberkholz/2013/03/25/programming-languages-ranked-by-expressiveness/

[16] *PyChecker: a python source code checking tool.* [Online; navštíveno 22.4.2017].
URL http://pychecker.sourceforge.net/

[17] *Pyflakes: Passive checker of Python programs.* [Online; navštíveno 24.4.2017].
URL https://pypi.python.org/pypi/pyflakes

[18] *Python 3 - Execution model.* [Online; navštíveno 24.4.2017].
URL https://docs.python.org/3.6/reference/executionmodel.html

[19] *Python AST Visualizer.* [Online; navštíveno 24.4.2017].
URL https://vpyast.appspot.com/

[20] *Python Official Website.* [Online; navštíveno 18.4.2017].
URL https://www.python.org/doc/essays/blurb/

[21] *Python rising popularity in stackoverflow trends.* [Online; navštíveno 18.4.2017].
URL https://stackoverflow.blog/2017/05/09/introducing-stack-overflow-trends/

[22] *Python Tutor - Visualize Python, Java, JavaScript, TypeScript, Ruby, C, and C++ code execution.* [Online; navštíveno 24.4.2017].
URL http://pythontutor.com/

[23] *Syntax and Semantics of Programming Languages.* [Online; navštíveno 29.4.2017].
URL http://homepage.divms.uiowa.edu/~slonnegr/plf/Book/Chapter1.pdf

[24] *Tokens, patterns and lexemes COMPILER DESIGN.* [Online; navštíveno 27.4.2017].
URL https://cnuinfotech-cd.blogspot.cz/2012/06/tokens-patterns-and-lexemes.html

[25] *Why Pylint is both useful and unusable, and how you can actually use it - Code Without Rules.* [Online; navštíveno 23.4.2017].
URL https://codewithoutrules.com/2016/10/19/pylint/

[26] *Špecifiácia dotazovacieho jazyka XPath.* [Online; navštíveno 27.4.2017].
URL https://www.w3.org/TR/xpath/

[27] Bui, T.: Analysis of Docker Security. *CoRR*, ročník abs/1501.02967, 2015.
URL http://arxiv.org/abs/1501.02967

[28] Chen, Z.; Chen, L.; Xu, B.: Hybrid Information Flow Analysis for Python Bytecode. In *2014 11th Web Information System and Application Conference*, Sept 2014, s. 95–100, doi:10.1109/WISA.2014.26.

[29] Cockburn, A.; Williams, L.: The costs and benefits of pair programming. *Extreme programming examined*, 2000: s. 223–247.

[30] Enbody, R. J.; Punch, W. F.; McCullen, M.: Python CS1 As Preparation for C++ CS2. *SIGCSE Bull.*, ročník 41, č. 1, Březen 2009: s. 116–120, ISSN 0097-8418, doi:10.1145/1539024.1508907.
URL http://doi.acm.org/10.1145/1539024.1508907

[31] Guo, P. J.: Online Python Tutor: Embeddable Web-based Program Visualization for Cs Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-1868-6, s. 579–584, doi:10.1145/2445196.2445368.
URL http://doi.acm.org/10.1145/2445196.2445368

[32] Jain, N.; Mangal, P.; Mehta, D.: AngularJS: A modern MVC framework in JavaScript. *Journal of Global Research in Computer Science*, ročník 5, č. 12, 2015: s. 17–23.

[33] Joseph Psotka, S. A. M.: *Intelligent Tutoring Systems: Lessons Learned*. Lawrence Erlbaum Associates, 1988, iSBN: 0805801928.

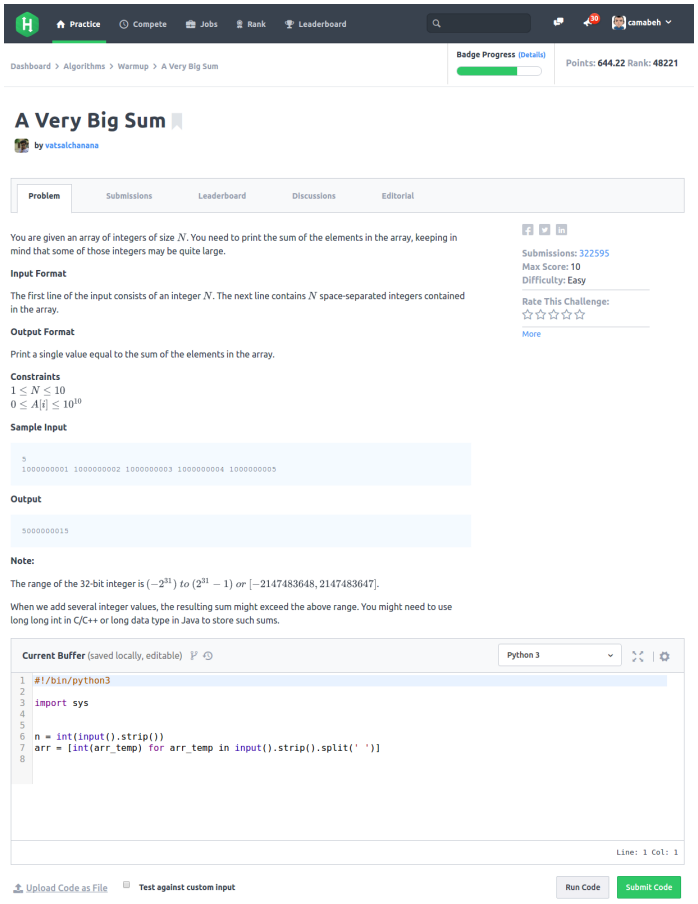
[34] Miller, W. L.; Baker, R. S.; Labrum, M. J.; aj.: Automated Detection of Proactive Remediation by Teachers in Reasoning Mind Classrooms. In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge*, LAK '15, New York, NY, USA: ACM, 2015, ISBN 978-1-4503-3417-4, s. 290–294, doi:10.1145/2723576.2723607.
URL http://doi.acm.org/10.1145/2723576.2723607

[35] Tang, T.; Rixner, S.; Warren, J.: An Environment for Learning Interactive Programming. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, New York, NY, USA: ACM, 2014, ISBN 978-1-4503-2605-6, s. 671–676, doi:10.1145/2538862.2538908.
URL http://doi.acm.org/10.1145/2538862.2538908

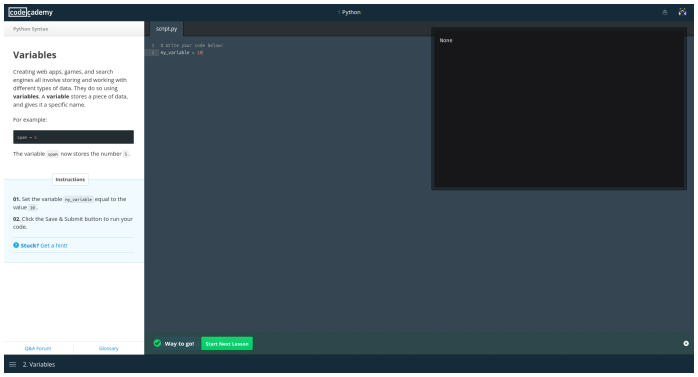
Prílohy

Príloha A

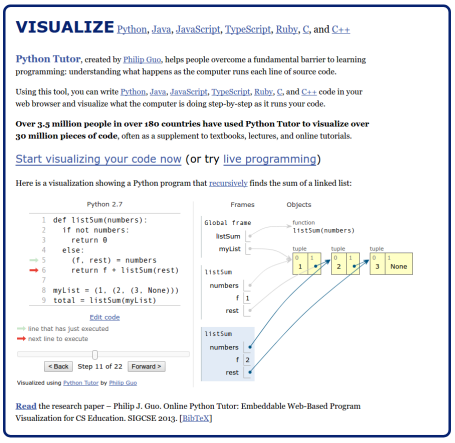
Obrázky alternatívnych riešení



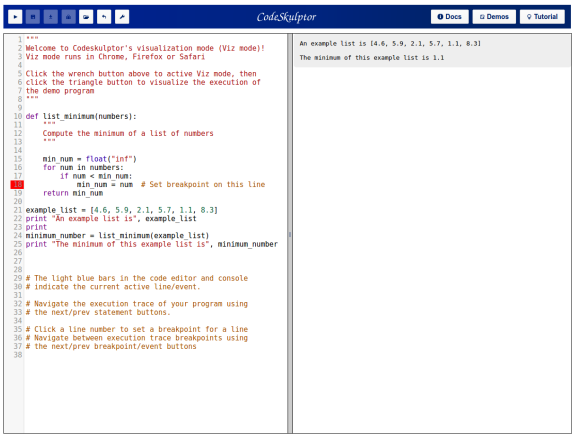
Obr. A.1: HackerRank prostredie trénovanie programovacích jazykov. Prevzaté z [9]



Obr. A.2: Codecademy. Prevzaté z [2]



Obr. A.3: Interaktívne prostredie Online Python Tutor. Prevzate z [22]



Obr. A.4: Interaktívne prostredie Code Skulptor. Prevzaté z [4]

Príloha B

Náležitosti potrebné pre nahranie novej úlohy do systému

Obsah súboru meta.json musí obsahovať tieto parametre

```
{
  "level": 3,
  "in": "file"
}
```

- *level*
  - 1: najľahšia obtiažnosť
  - 2: stredná obtiažnosť
  - 3: najvyššia obtiažnosť
- *in*
  - stream: vstupom skriptu je štandardný vstup
  - file: v aktuálnom adresári testovaného skriptu sa použije ako vstup súbor `file.txt`

Testy

Testy musia byť pomenované vo formáte `test.N.[in|out]`, kde N >= 1. A podmiemkou je že, musia byť minimálne `test.1.in` a `test.1.out`. V prípade použitia ako štandardný vstup do skriptu, sa `test.N.in` nastaví ako jeho vstup, v prípade použitia súboru sa nakopíruje do adresára testovaného skriptu pod menom `file.txt`.

Súbor problem.html musí obsahovať

```
<h2 id="title">Fibonacci</h2>
<p id="description">Calculate n-th fibonacci number</p>

<h4>INPUT:</h4>

<pre>
12
</pre>

<h4>EXPECTED OUTPUT:</h4>

<pre>
144
</pre>
```

Tento súbor musí obsahovať html elementy s identifikátorom `title` a `description`. Tieto hodnoty sa počas nahrávania vyexportujú a uložia do databázy ako názov a popis úlohy.

Príloha C

Obsah súborov na disku